Mehdi Lakbar
M1 EEA

Rapport de mini-projet : JEU HEX

Le jeu Hex est assez facile : le joueur bleu doit relier les deux côtés bleus et le joueur rouge doit faire de même avec les côtés rouges. La spécificité de ce jeu est qu'il y'a forcément un gagnant, donc pas de partie nulle.

Pour la reconnaissance des cases, il faut d'abord relever le centre de chaque Hexagone du plateau. On stocke chaque centre dans un tableau de structures de type Point. Chaque objet de type point (qui correspond donc à une case du plateau) contient une coordonnée X, une coordonnée Y, une couleur (Rouge, Bleu ou Blanc ou R/B/W) et les coordonnées de ses 6 voisins (pour la reconnaissance du gagnant).

Une fois qu'on a relevé le centre de chaque hexagone, on les rend cliquables grâce à une simple recherche de l'hexagone le plus proche d'où on a cliqué, par la méthode des moindres carrés. A chaque clic sur l'écran, on parcourt notre tableau de structures de types Points, et on cherche donc celui dont la distance est la plus petite. Il faut cependant faire attention à ce que les cases que l'on a choisies n'aient pas déjà une couleur. Si c'est le cas (leur attribut couleur est différent de W=white), alors cette case n'est pas cliquable et on choisit la case la plus proche qui a une couleur W.

On peut alors tracer les hexagones pleins voulus (à tour de rôle rouge, puis bleu), à l'aide des fonctions drawLine() et fillPolygon(). La structure de points ici pose problème car la fonction fillPolygon() utilise également un pointeur vers un tableau de structures de points qui ne correspond pas à la nôtre. Il faut donc modifier le code de la fonction void fillPolygon() dans le fichier « stm32746g_discovery_lcd.c » pour la rendre compatible avec notre structure de points.

Le jeu est jouable, il ne reste plus qu'à déterminer le gagnant. Cela se ferait par une recherche exhaustive. Pour chercher si le gagnant est rouge par exemple, on commencerait par les cases sur les bords, et on regarderait pour chacun de ses voisins si sa couleur est rouge, puis on regarderait les voisins du voisins… jusqu'à tomber (ou non) sur une case du bord opposé. Si c'est le cas, alors le gagnant est bien le rouge.

Pour le menu pause, on commence par rendre cliquable le coin en haut à doitre du plateau (dans la tâche de jeu), en encadrant TS_State.touchX[0] et TS_State.touchY[0]. Les tâches de jeu et de pause communiquent par blackboard. Si la variable globale pause est à 1, alors la tâche de jeu est suspendue et celle de pause active. L'inverse est valable si pause==0.

Pour afficher le menu pause ou le signe Game Over, on utilise la transparence, qui permet un rendu bien plus joli. Pour cela, on utilise les fonctions suivantes : BSP_LCD_SetLayerVisible() (on spécifie l'index de la couche voulue et ENABLE ou DISABLE en fonction des cas) et BSP_LCD_SetTransparency(int indexLayer, int blendingFactor). Il faut que blendingFactor ait une valeur entre 0x00 et 0xFF. Plus ce dernier est grand, plus la couche sélectionnée sera transparente. A chaque activation de la tâche pause par exemple, la tâche jeu est suspendue et la deuxième couche est activée avec un facteur de transparence blendingFactor = 0x6F (j'ai choisi cette valeur car il s'agit de celle qui rendait le mieux visuellement avec les couleurs du menu pause et celles du plateau).

J'ai fini par activer le DMA et le DAC tout simplement en suivant les étapes spécifiées dans le fichier DAC disponible sur eCampus.